Tutorial 9

Greedy Algorithms 2

CSC 311 (Fall 2018)

Coin changing problem
Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer.

a- Describe a greedy algorithm to make change.
b- Solve the following instance from coin change problem based on the algorithm.

Denom{ 1, 2, 5, 10, 20, 50, 100, 500, 1000}.
A= 234
A=8
A=3721

Solution:

a- Algorithm

Input Parameters: *denom*,*A* //*denom*[1] > *denom*[2] > ⋯ > *denom*[*n*] = 1.
Output Parameters: None

```
greedy_coin_change(denom,A) {
        i = 1
        while (A > 0) {
                c = A/denom[i]
                println("use " + c + " coins of denomination " +
                                denom[i])
                A = A - c * denom[i]
                i = i + 1
        }
}
```

b- .

I	1	2	3	4	5	6	7	8	9
Denom	1000	500	100	50	20	10	5	2	1
C	0	0	2	0	1	1	0	2	
A	234	234	34	34	14	4	4	0	

I	1	2	3	4	5	6	7	8	9
Denom	1000	500	100	50	20	10	5	2	1
C	0	0	0	0	0	0	1	1	1
A	8	8	8	8	8	8	3	1	0

I	1	2	3	4	5	6	7	8	9
Denom	1000	500	100	50	20	10	5	2	1
C	3	1	1	0	1	0	0	0	1
A	721	221	21	21	1	1	1	1	0

Tutorial 9

Greedy Algorithms 2

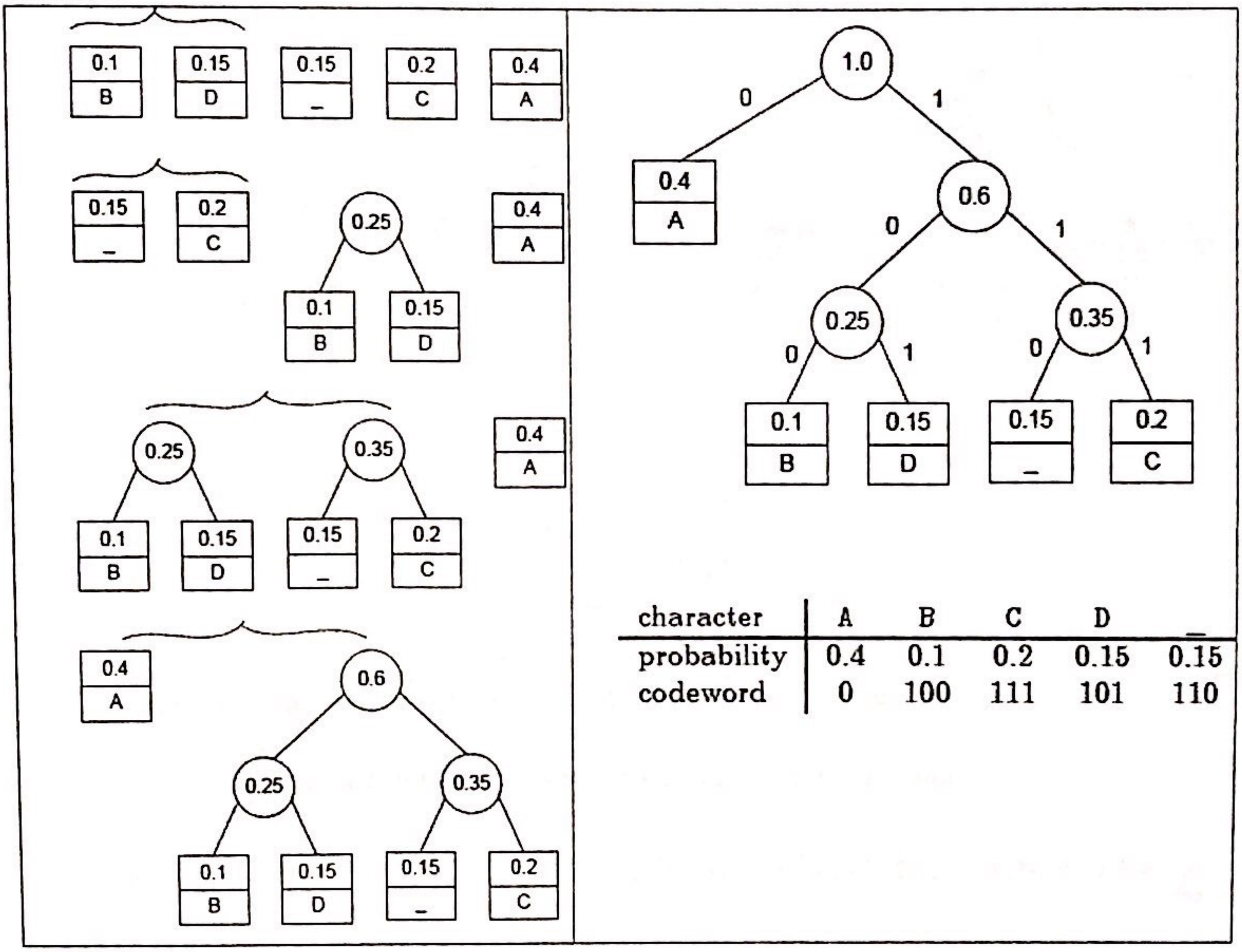
CSC 311 (Fall 2018)

1- Consider the following data:

symbol	A	B	C	D	_
frequency	0.4	0.1	0.2	0.15	0.15

a. Construct a Huffman code.

Solution:



character	A	B	C	D	_
probability	0.4	0.1	0.2	0.15	0.15
codeword	0	100	111	101	110

b. Encode ABACABAD using the code of question (a).

0100011101000101

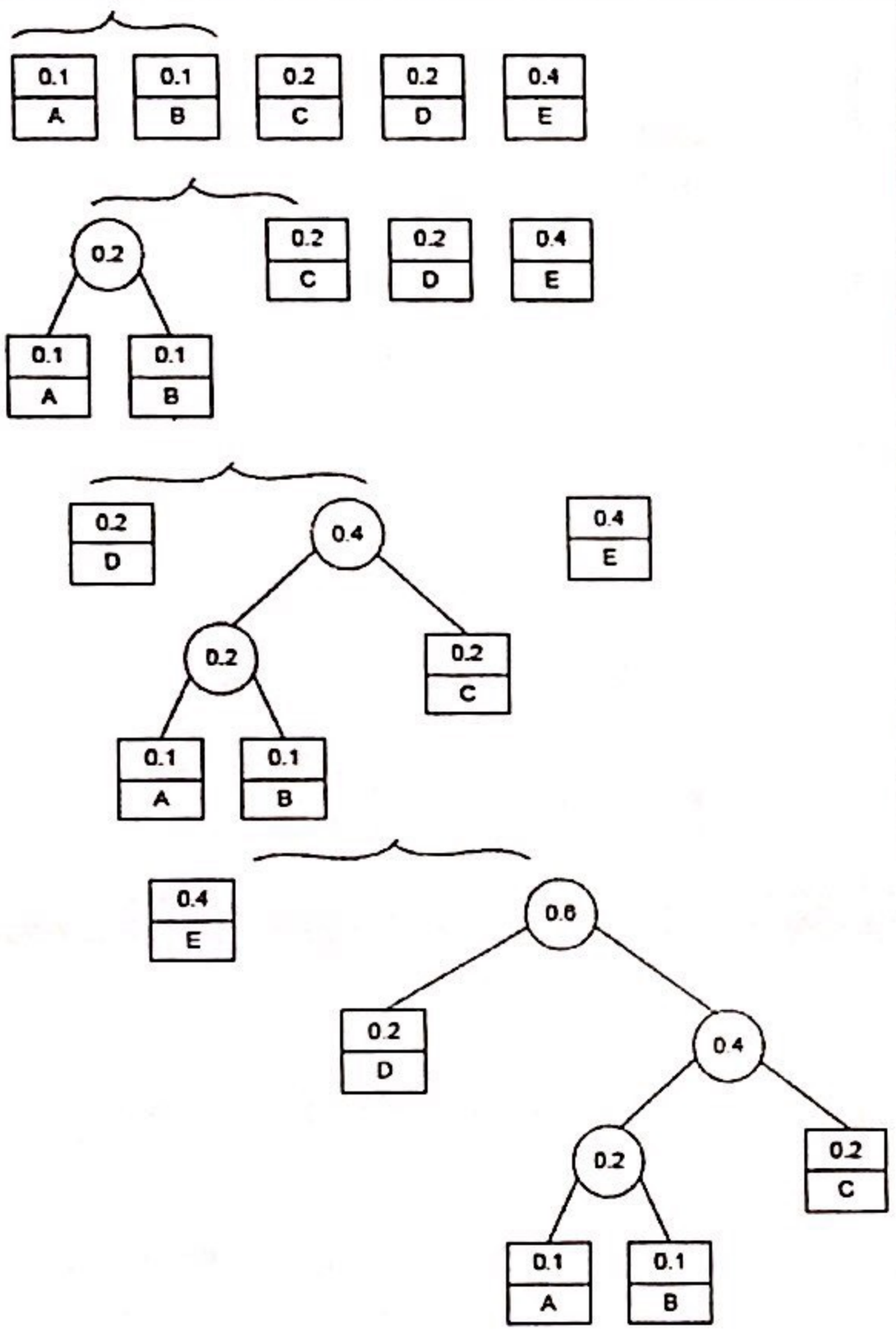
c. Decode 100010111001010 using the code of question (a).

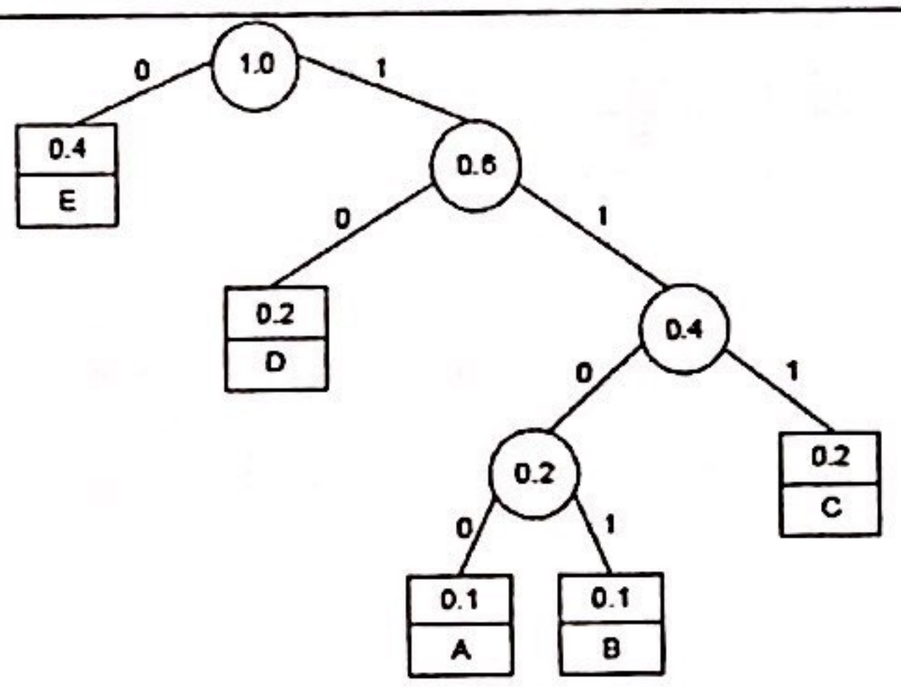
100|0|101|110|0|101|0
B A D _ A D A

2- For data transmission purposes, it is often desirable to have a code with a minimum variance of the codeword lengths (among codes of the same average length). Compute the average and variance of the codeword length in two Huffman codes that result from a different tie breaking during a Huffman code construction for the following data:

symbol	A	B	C	D	E
frequency	0.1	0.1	0.2	0.2	0.4

Solution:





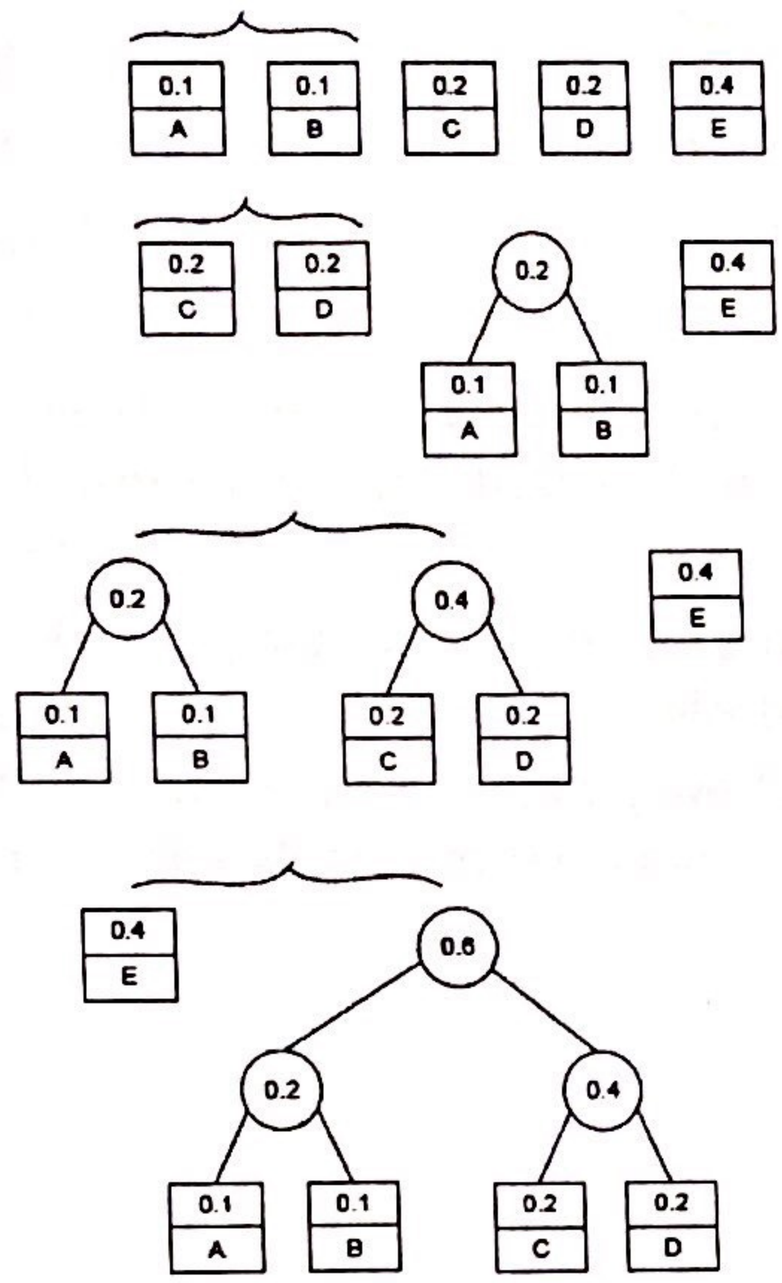
character	A	B	C	D	E
probability	0.1	0.1	0.2	0.2	0.4
codeword	1100	1101	111	10	0
length	4	4	3	2	1

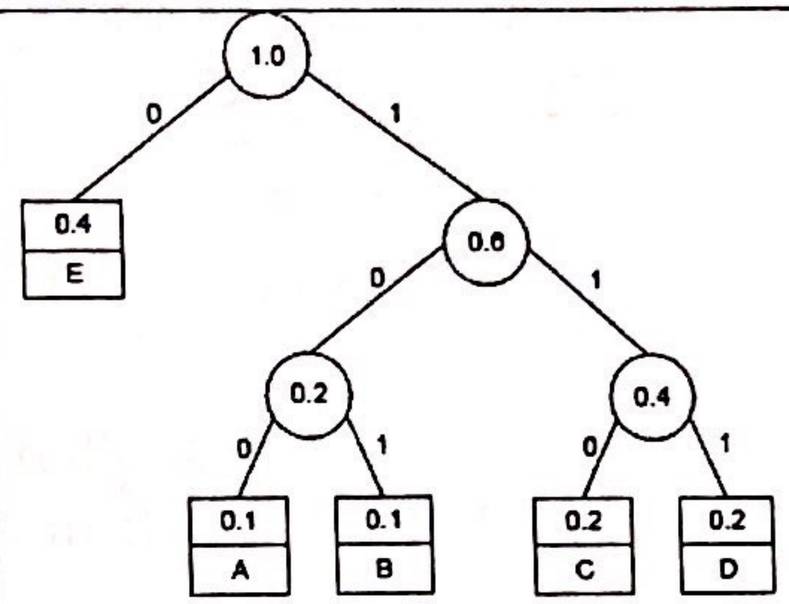
Thus, the mean and variance of the codeword's length are, respectively,

$$\bar{l} = \sum_{i=1}^{5} l_i p_i = 4 \cdot 0.1 + 4 \cdot 0.1 + 3 \cdot 0.2 + 2 \cdot 0.2 + 1 \cdot 0.4 = 2.2 \quad \text{and}$$

$$Var = \sum_{i=1}^{5} (l_i - \bar{l})^2 p_i = (4\text{-}2.2)^2 0.1 + (4\text{-}2.2)^2 0.1 + (3\text{-}2.2)^2 0.2 + (2\text{-}2.2)^2 0.2 + (1\text{-}2.2)^2 0.4 = 1.36.$$

Here is another way:





character	A	B	C	D	E
probability	0.1	0.1	0.2	0.2	0.4
codeword	100	101	110	111	0
length	3	3	3	3	1

Thus, the mean and variance of the codeword's length are, respectively,

$$\bar{l} = \sum_{i=1}^{5} l_i p_i = 2.2 \quad \text{and} \quad Var = \sum_{i=1}^{5} (l_i - \bar{l})^2 p_i = 0.96.$$

3- From the following list, someone is allowed to carry one bag with him and the bag holds no more than **16 pounds,** What should he be putting in the bag?

Item	Weight	value
A	8	20
B	10	25
C	2	8
D	4	12
E	1	5
F	4	6
G	1	8

Solution:

item	weight	value	sum
G	1	8	1
E	1	5	2
C	2	8	4
D	4	12	8
A	8	20	16

- Calculate the value-per-pound $\rho_i = \frac{v_i}{w_i}$ for $i = 1, 2, \ldots, n$.
- Sort the items by decreasing ρ_i.
  Let the sorted item sequence be $1, 2, \ldots, i, \ldots n$, and the corresponding value-per-pound and weight be ρ_i and w_i respectively.

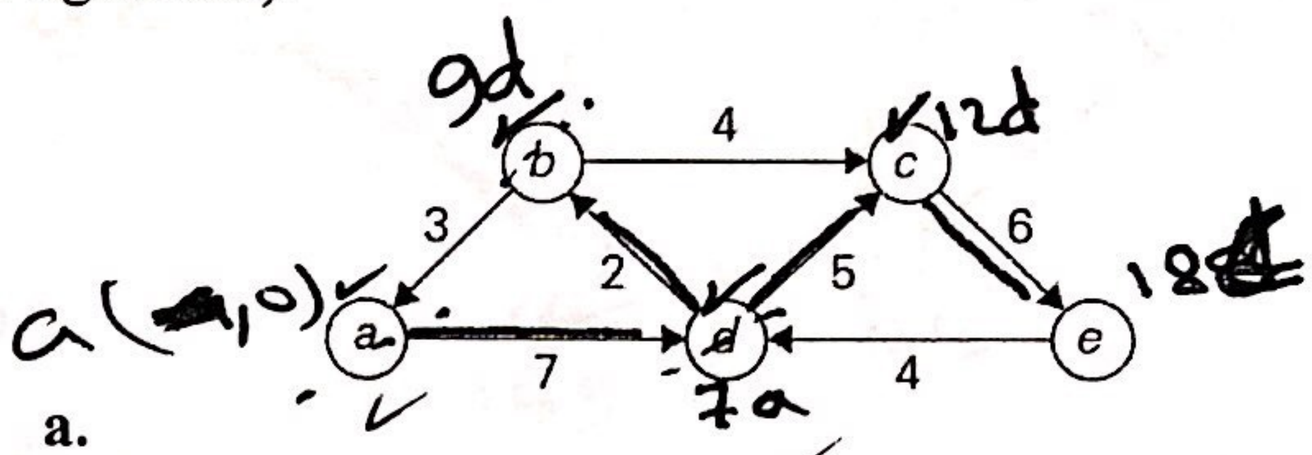
- Let k be the current weight limit (Initially, $k = K$).
  In each iteration, we choose item i from the head of the unselected list.
  - If $k \geq w_i$, set $x_i = 1$ (we take item i), and reduce $k = k - w_i$, then consider the next unselected item.
  - If $k < w_i$, set $x_i = k/w_i$ (we take a fraction k/w_i of item i), Then the algorithm terminates.

Tutorial 9

Greedy Algorithms

CSC 311 (Fall 2018)

1- Solve the following instances of the single-source shortest-paths problem with vertex a as the source (Dijkstra's Algorithm):



a.

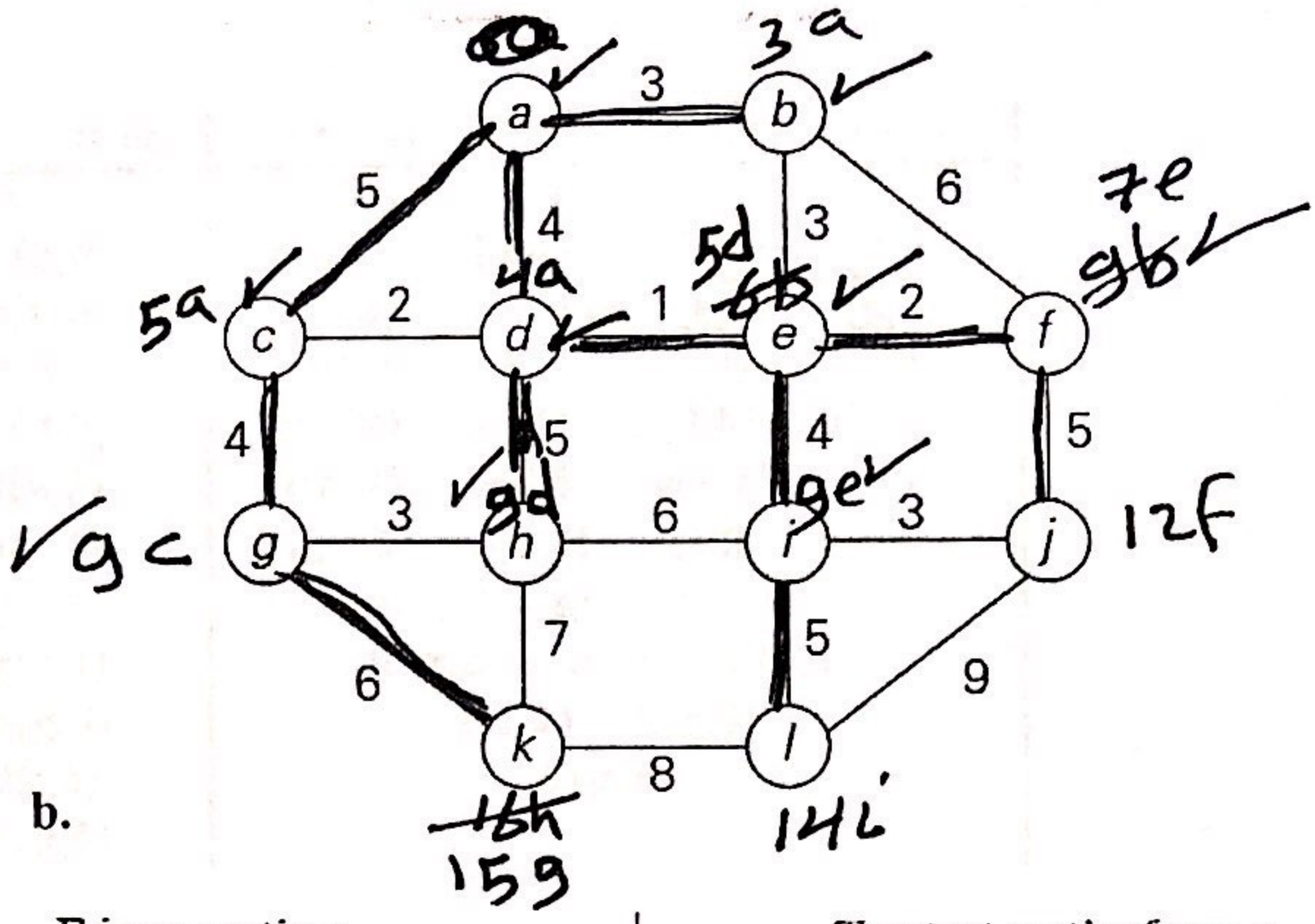
Solution:

dist[s] ←0 *(distance to source vertex is zero)*
for all v ∈ V–{s}
do dist[v] ←∞ *(set all other distances to infinity)*
S←∅ *(S, the set of visited vertices is initially empty)*
Q←V *(Q, the queue initially contains all vertices)*
while Q ≠ ∅ *(while the queue is not empty)*
do u ← mindistance(Q,dist) *(select the element of Q with the min. distance)*
S←S∪{u} *(add u to list of visited vertices)*
for all v ∈ neighbors[u]
do if dist[v] > dist[u] + w(u, v) *(if new shortest path found)*
then d[v] ←d[u] + w(u, v) *(set new value of shortest path)*
(if desired, add traceback code)
return dist

Tree vertices	Remaining vertices
a(-,0)	b(-,∞) c(–,∞) d(a,7) e(-,∞)
d(a,7)	b(d,7+2) c(d,7+5) e(-,∞)
b(d,9)	c(d,12) e(-,∞)
c(d,12)	e(c,12+6)
e(c,18)	

The shortest paths (identified by following nonnumeric labels backwards from a destination vertex to the source) and their lengths are:

from a to d: $a - d$ of length 7
from a to b: $a - d - b$ of length 9
from a to c: $a - d - c$ of length 12
from a to e: $a - d - c - e$ of length 18



b.

Solution:

Tree vertices	Fringe vertices	Shortest paths from a
a(-,0)	**b(a,3)** c(a,5) d(a,4)	to b: $a-b$ of length 3
b(a,3)	c(a,5) **d(a,4)** e(b,3+3) f(b,3+6)	to d: $a-d$ of length 4
d(a,4)	**c(a,5)** e(d,4+1) f(a,9) h(d,4+5)	to c: $a-c$ of length 5
c(a,5)	**e(d,5)** f(a,9) h(d,9) g(c,5+4))	to e: $a-d-e$ of length 5
e(d,5)	**f(e,5+2)** h(d,9) g(c,9) i(e,5+4)	to f: $a-d-e-f$ of length 7
f(e,7)	**h(d,9)** g(c,9) i(e,9) j(f,7+5)	to h: $a-d-h$ of length 9
h(d,9)	**g(c,9)** i(e,9) j(f,12) k(h,9+7))	to g: $a-c-g$ of length 9
g(c,9)	**i(e,9)** j(f,12) k(g,9+6)	to i: $a-d-e-i$ of length 9
i(e,9)	**j(f,12)** k(g,15) l(i,9+5)	to j: $a-d-e-f-j$ of length 12
j(f,12)	k(g,15) **l(i,14)**	to l: $a-d-e-i-l$ of length 14
l(i,14)	**k(g,15)**	to k: $a-c-g-k$ of length 15
k(g,15)		

2- Apply Prim's algorithm to the graph in paragraph (b) of the previous question.

Solution:

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
while (Q not empty)
    u = ExtractMin(Q);
    for each v ∈ Adj[u]
        if (v ∈ Q and w(u,v) < key[v])
            p[v] = u;
            key[v] = w(u,v);
```

Tree vertices	Priority queue of fringe vertices
a(-,-)	b(a,3) c(a,5) d(a,4)
b(a,3)	c(a,5) d(a,4) e(b,3) f(b,6)
e(b,3)	c(a,5) d(e,1) f(e,2) i(e,4)
d(e,1)	c(d,2) f(e,2) i(e,4) h(d,5)
c(d,2)	f(e,2) i(e,4) h(d,5) g(c,4)
f(e,2)	i(e,4) h(d,5) g(c,4) j(f,5)
i(e,4)	h(d,5) g(c,4) j(i,3) l(i,5)
j(i,3)	h(d,5) g(c,4) l(i,5)
g(c,4)	h(g,3) l(i,5) k(g,6)
h(g,3)	l(i,5) k(g,6)
l(i,5)	k(g,6)
k(g,6)	

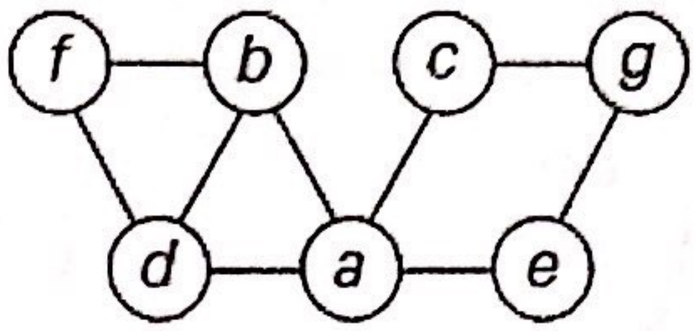
The minimum spanning tree found by the algorithm comprises the edges *ab*, *be*, *ed*, *dc*, *ef*, *ei*, *ij*, *cg*, *gh*, *il*, *gk*.

Tutorial 8

Graph Algorithms

CSC 311 (Fall 2018)

1- Consider the following graph.



a- Write down the adjacency matrix and adjacency lists specifying this graph. (Assume that the matrix rows and columns and vertices in the adjacency lists follow in the alphabetical order of the vertex labels.)

b- Starting at vertex *a* and resolving ties by the vertex alphabetical order, traverse the graph by breadth-first search and construct the corresponding depth-first search tree.

c- Starting at vertex *a* and resolving ties by the vertex alphabetical order, traverse the graph by depth-first search and construct the corresponding depth-first search tree. Give the order in which the vertices were reached for the first time (pushed onto the traversal stack) and the order in which the vertices became dead ends (popped off the stack).

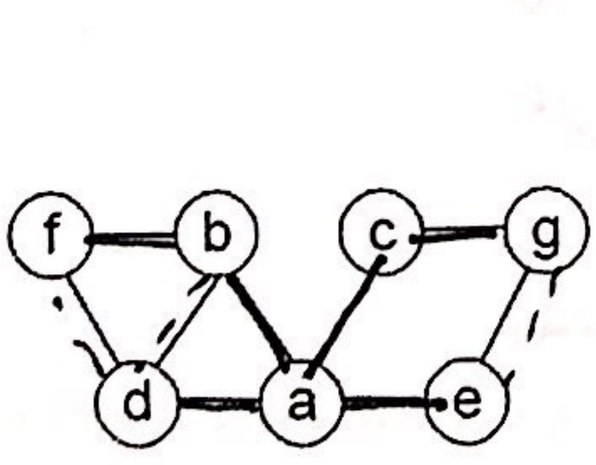
Solution:

$$
\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array}
\begin{array}{c}
\begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\
\left[\begin{array}{ccccccc}
0 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0
\end{array}\right]
\end{array}
$$

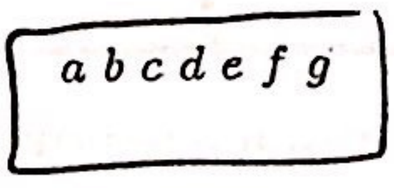
a	$\rightarrow b \rightarrow c \rightarrow d \rightarrow e$
b	$\rightarrow a \rightarrow d \rightarrow f$
c	$\rightarrow a \rightarrow g$
d	$\rightarrow a \rightarrow b \rightarrow f$
e	$\rightarrow a \rightarrow g$
f	$\rightarrow b \rightarrow d$
g	$\rightarrow c \rightarrow e$

a-

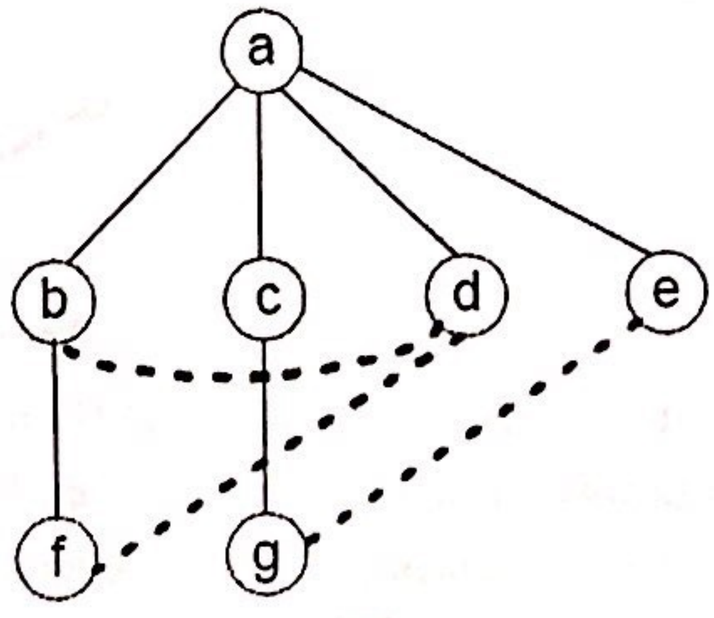
b- the graph; (ii) the traversal's queue; (iii) the tree (the tree and cross edges are shown with solid and dotted lines, respectively).



(i)



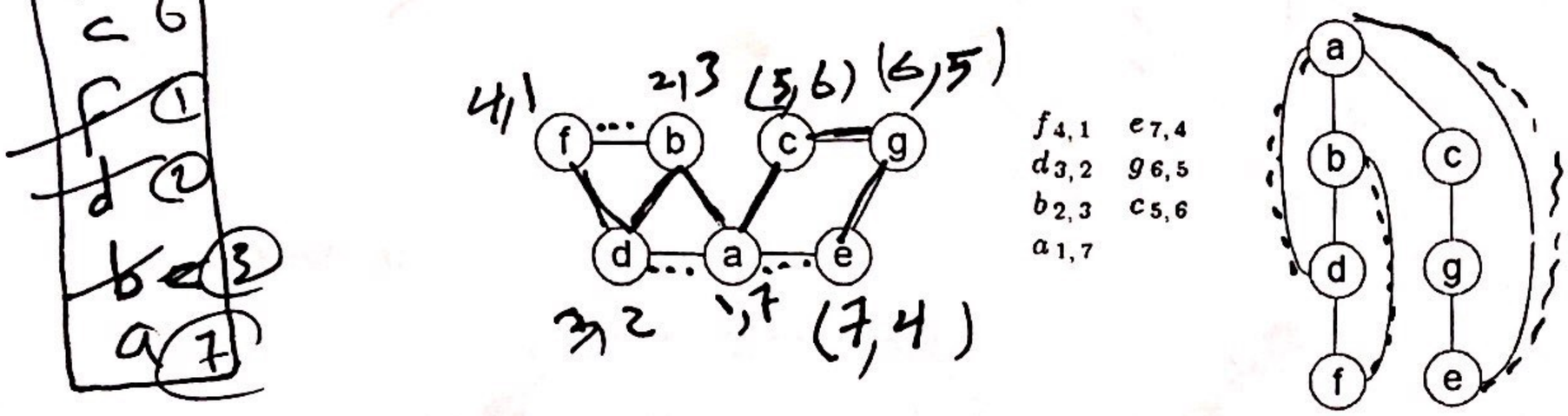
(ii)



(iii)

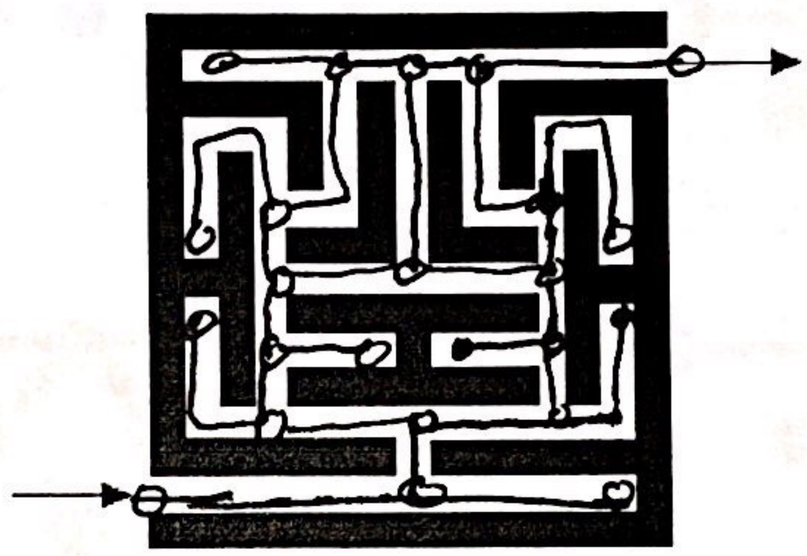
c- See below: (i) the graph; (ii) the traversal's stack (the first subscript number indicates the order in which the vertex was visited, i.e., pushed onto the stack, the second one

indicates the order in which it became a dead-end, i.e., popped off the stack); (iii) the DFS tree (with the tree edges shown with solid lines and the back edges shown with dashed lines).

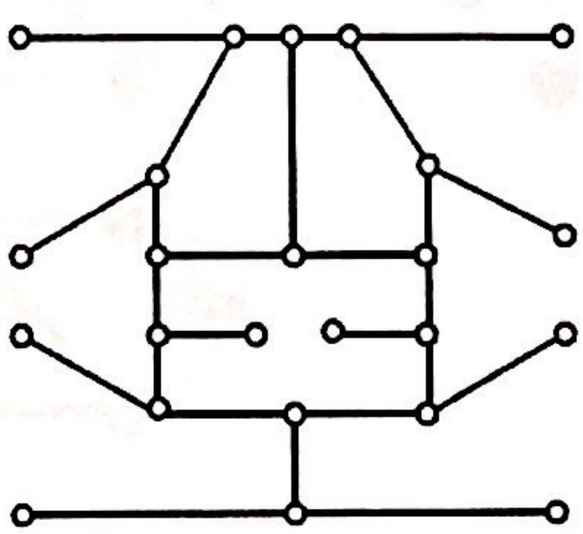


(i) (ii) (iii)

2- Construct such a graph for the following maze.



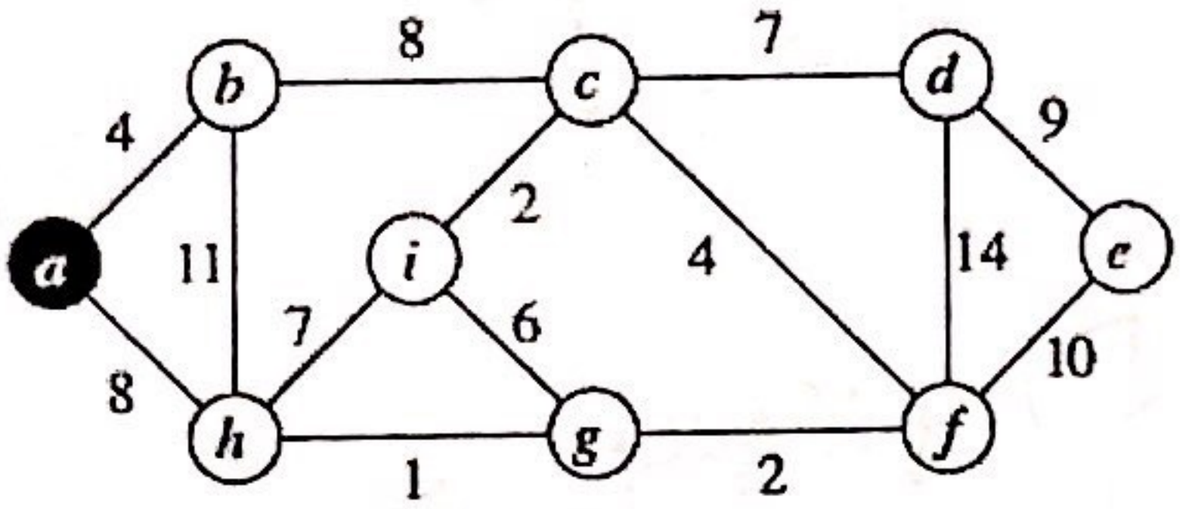
Which traversal—DFS or BFS—would you use if you found yourself in a maze and why?
Solution:



DFS is much more convenient for going through a maze than BFS. When DFS moves to a next vertex, it is connected to a current vertex by an edge (i.e., "close nearby" in the physical maze), which is not generally the case for BFS. In fact, DFS can be considered a generalization of an ancient right-hand rule for maze traversal: go through the maze in such a way so that your right hand is always touching a wall.

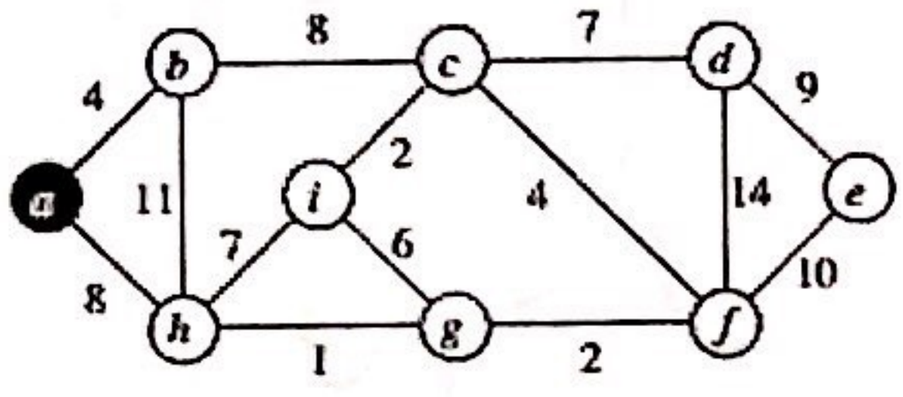
3- Given the following weighted graph, find a *spanning tree*

Prime

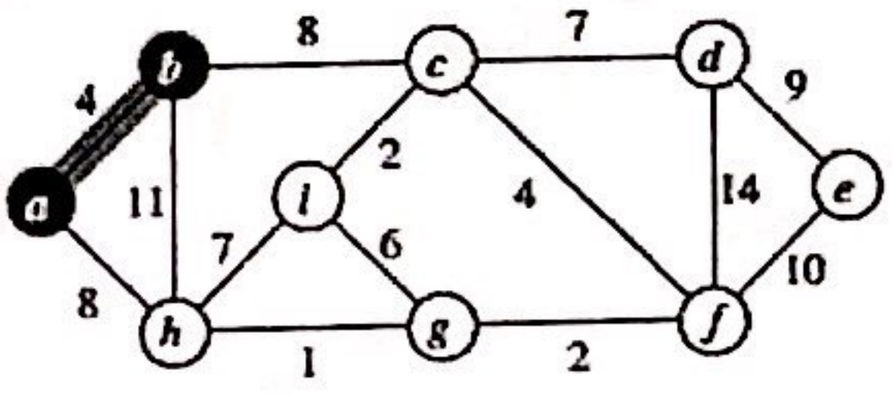


Solution:

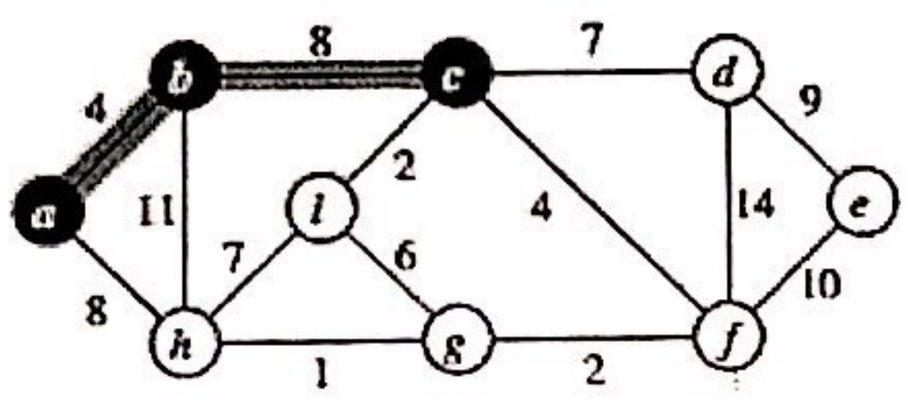
(a)



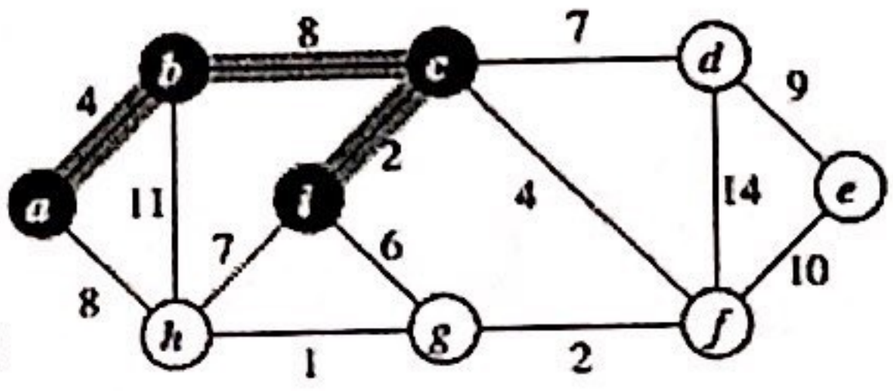
(b)



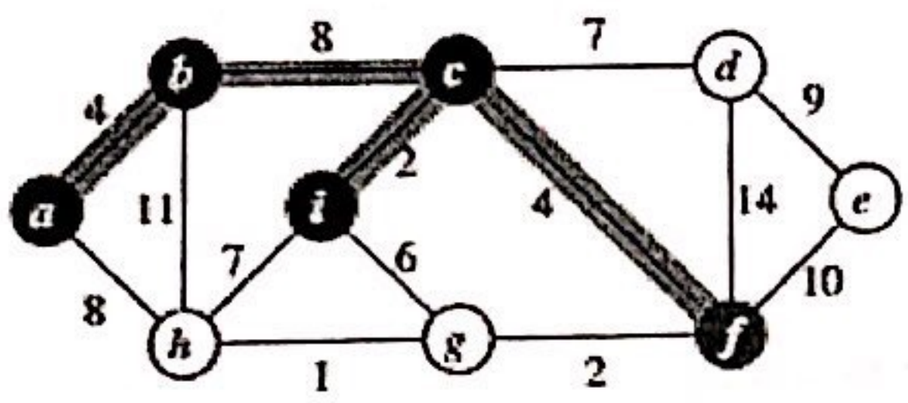
(c)



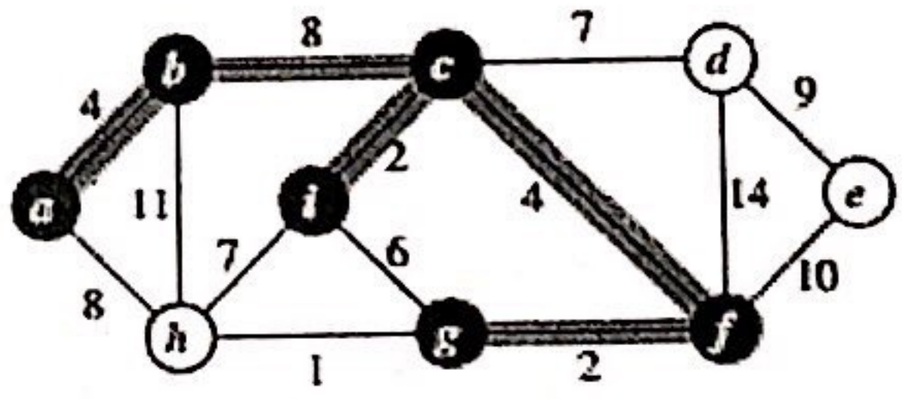
(d)



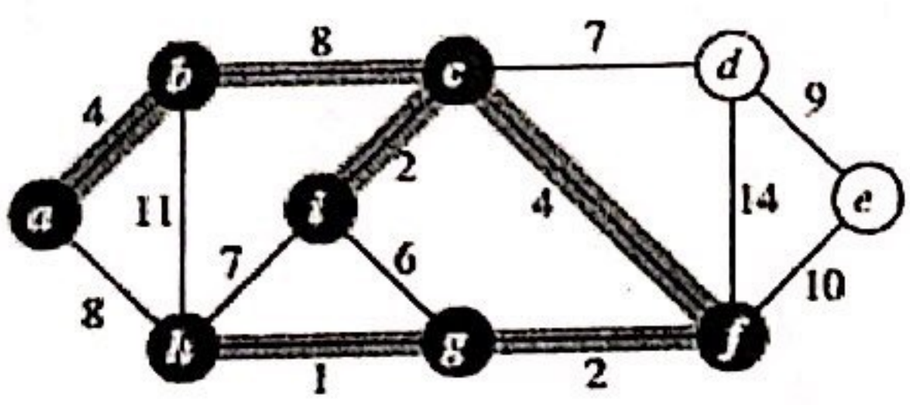
(e)



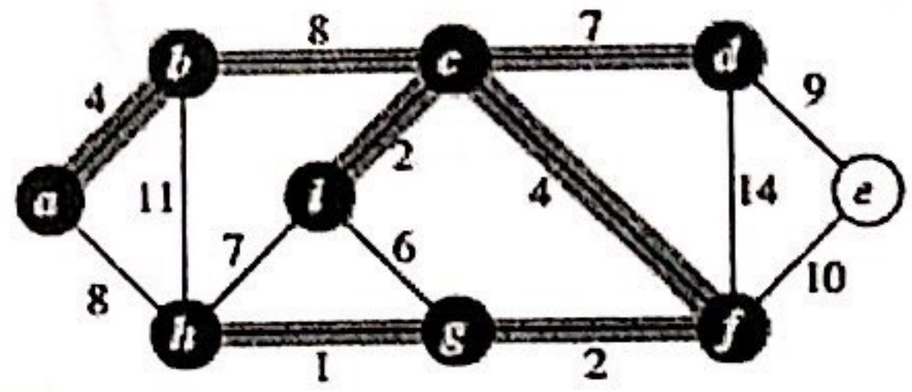
(f)



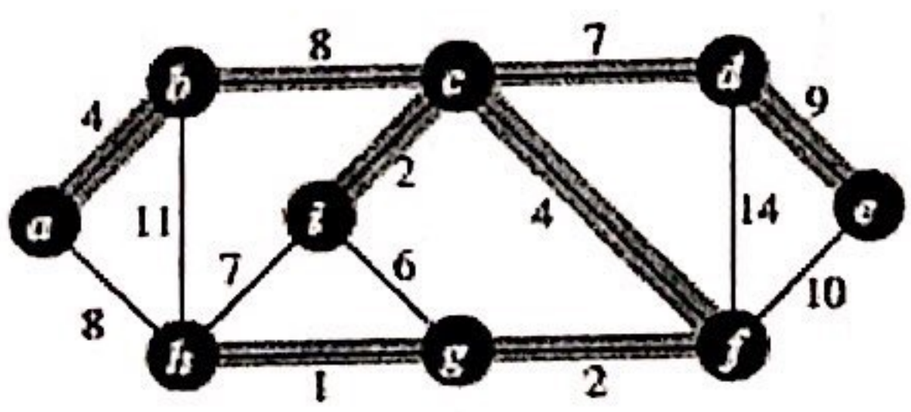
(g)



(h)



(i)



max $1x_1 + 4x_2 + 3x_3$

v_1 or $b_1 = 1$

$b_2 = 4$

$b_3 = 3$

such that $1x_1 + 4x_2 + 3x_3 \leq 4$

$w_1 = 1$

$w_2 = 4$

$w_3 = 3$

$w_{max} = 4$

$x_j = 0$ or 1 for $j = 1, 2, 3$

Items	x_1	x_2	x_3	≤ 4
w_i	1	4	3	
b_i	1	4	3	

Tutorial 6

Dynamic Programing

CSC 311 (Fall 2018)

1- Design an efficient algorithm for computing the binomial coefficient C(n, k) that uses no multiplications. This coefficient can be calculated using the following function:

$$C(n, k) = C(n-1, k-1) + C(n-1, k), \quad n >= k >= 0$$

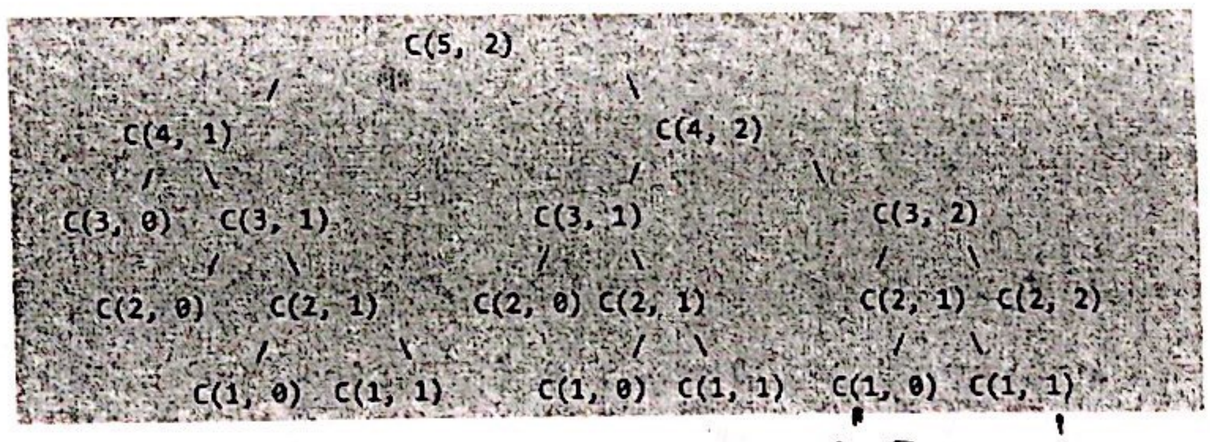
$$C(n, 0) = C(n, n) = 1, \quad n >= 0$$

a- Solve the problem using simple recursive algorithm and using dynamic programing.

b- Solve it for C(5,2)=10

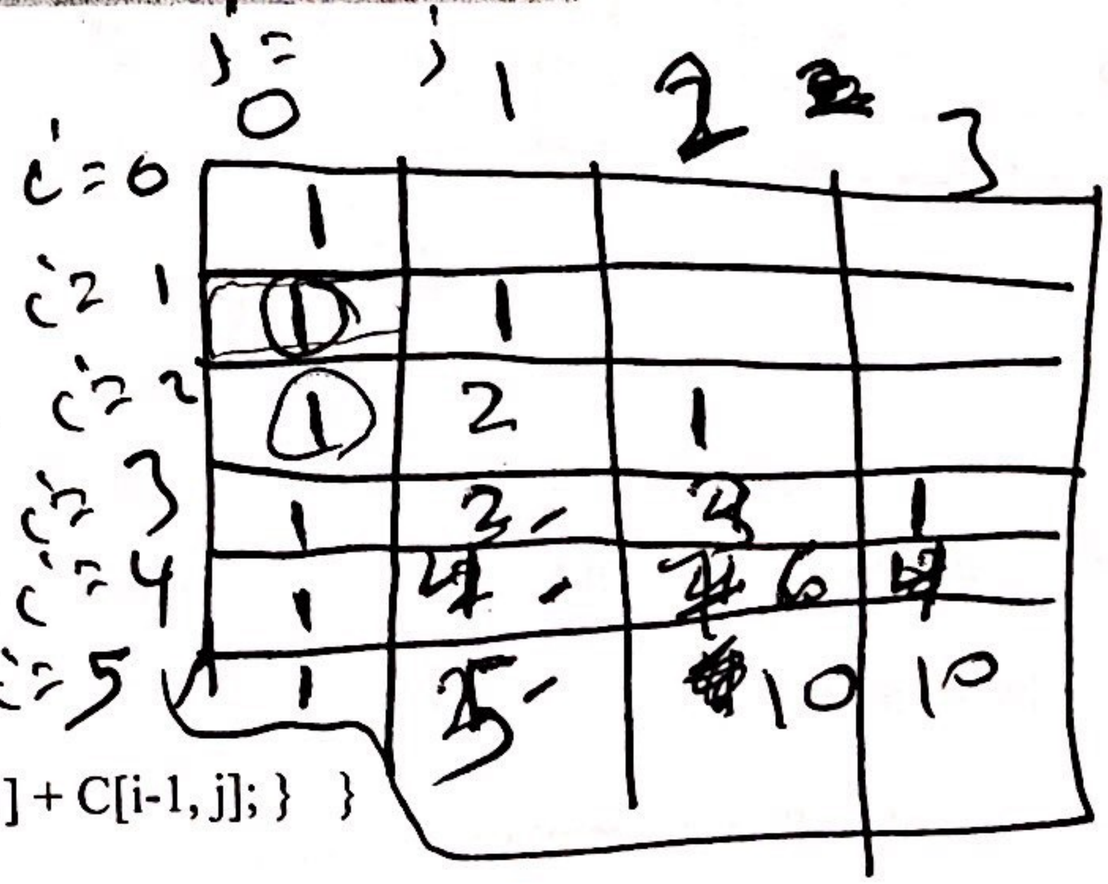
```
binomialCoeff(n, k) {
if (k==0 || k==n)
  return 1;
return  binomialCoeff(n-1, k-1) + binomialCoeff(n-1, k);}
```

(computes the same subproblems again and again)!!



By DP:

```
binomialCoeff(n, k)
for (i = 0 : n) {
    for (j = 0:  min(i, k) ) {
        if (j == 0 || j == i)
            C[i,  j] = 1;
        else
            C[i , j] = C[i-1,j-1] + C[i-1, j]; }   }
  return C; }
```



O(n*k)

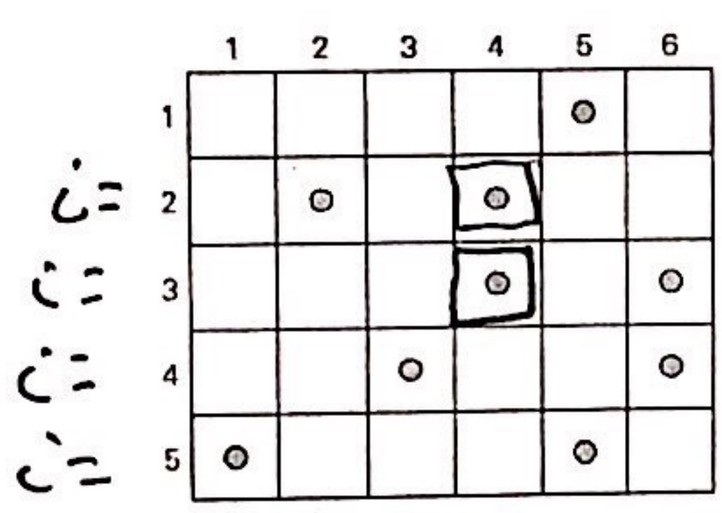
2- Applies dynamic programming to compute the largest number of coins a robot can collect on an n × m board by starting at (1, 1) and moving right and down from upper left to down right corner.

Solution:

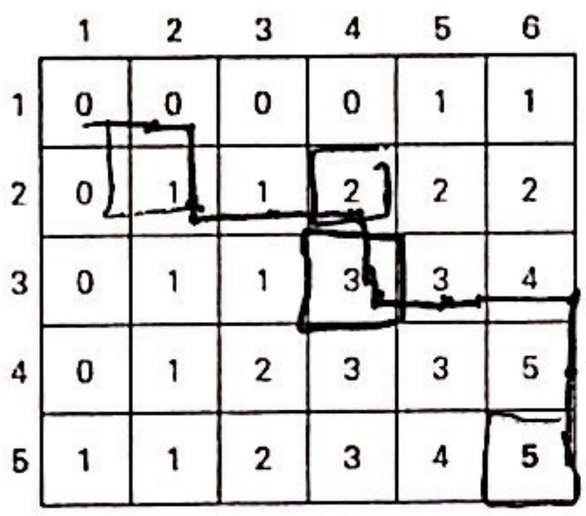
$$F(i, j) = \max\{F(i-1, j), F(i, j-1)\} + c_{ij} \quad \text{for } 1 \le i \le n, \quad 1 \le j \le m$$
$$F(0, j) = 0 \quad \text{for } 1 \le j \le m \quad \text{and} \quad F(i, 0) = 0 \quad \text{for } 1 \le i \le n,$$

ALGORITHM *RobotCoinCollection*$(C[1..n, 1..m])$

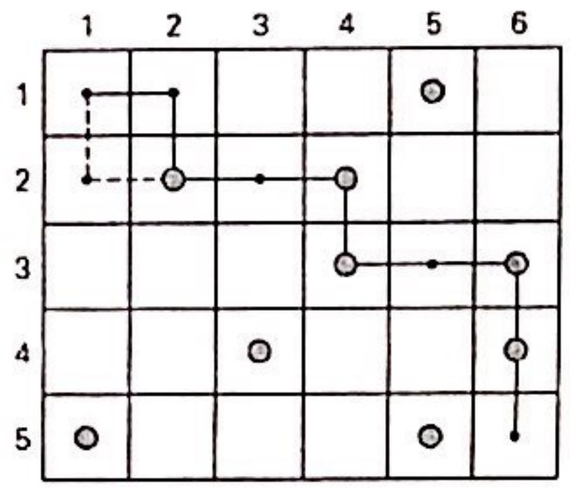
```
//Input: Matrix C[1..n, 1..m] whose elements are equal to 1 and 0
//for cells with and without a coin, respectively
//Output: Largest number of coins the robot can bring to cell (n, m)
F[1, 1] ← C[1, 1]; for j ← 2 to m do F[1, j] ← F[1, j − 1] + C[1, j]
for i ← 2 to n do
    F[i, 1] ← F[i − 1, 1] + C[i, 1]
    for j ← 2 to m do
        F[i, j] ← max(F[i − 1, j], F[i, j − 1]) + C[i, j]
return F[n, m]
```



C



F



Two possible paths

$\Theta(nm)$

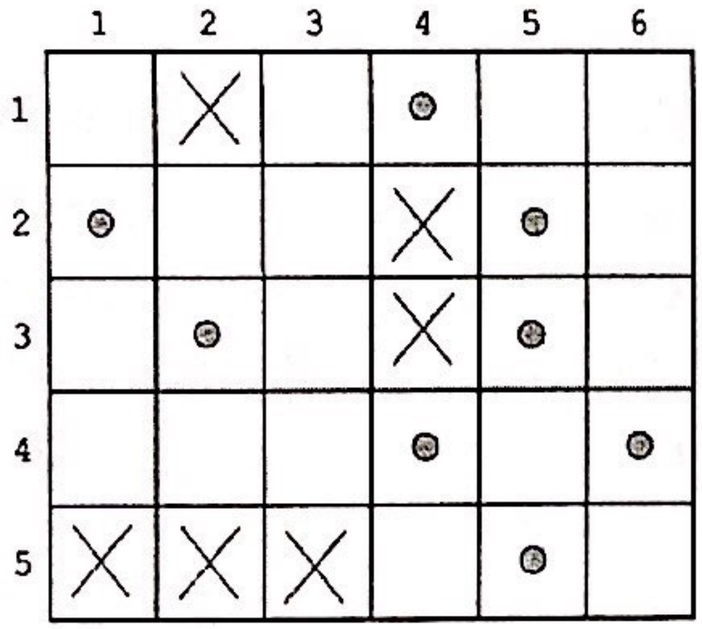
Tracing the computations backward makes it possible to get an optimal path:

- if $F(i-1, j) > F(i, j-1)$, an optimal path to cell (i, j) must come down from the adjacent cell above it;
- if $F(i-1, j) < F(i, j-1)$, an optimal path to cell (i, j) must come from the adjacent cell on the left;
- and if $F(i-1, j) = F(i, j-1)$, it can reach cell (i, j) from either direction.

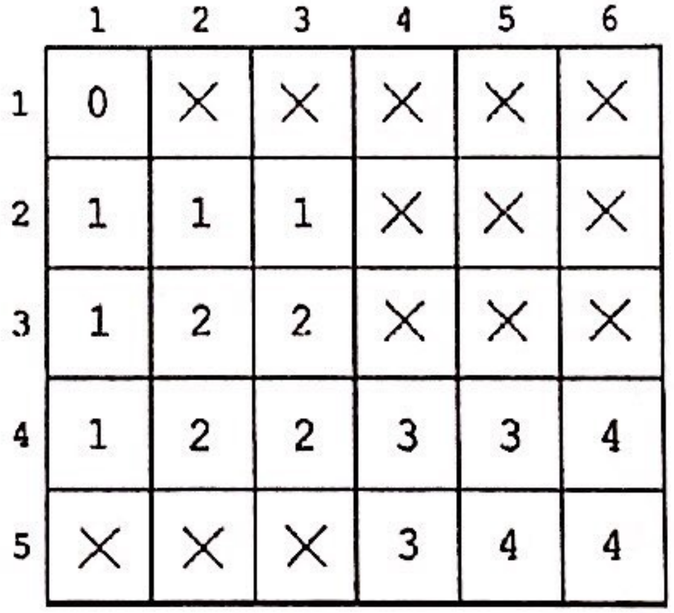
3- How would you modify the dynamic programming algorithm for the coin-collecting problem (previous exercise) if some cells on the board are inaccessible for the robot? Apply your algorithm to the board below, where the inaccessible cells are shown by X's. How many optimal paths are there for this board?

Solution:

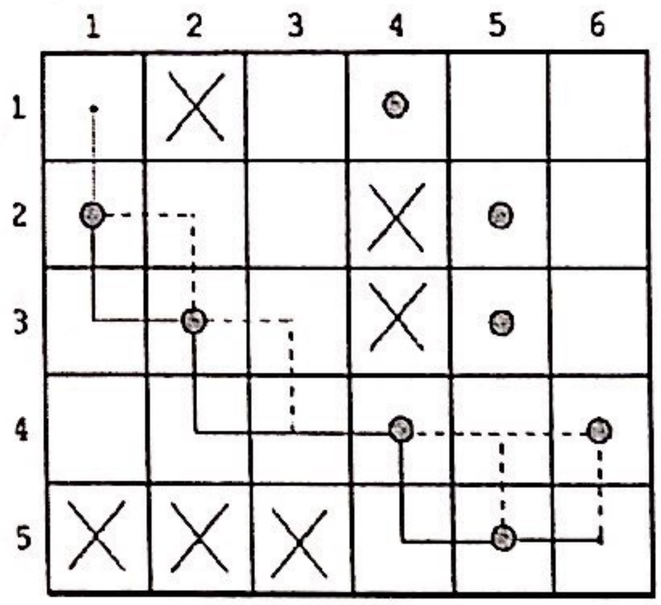
If a cell is inadmissible or has no admissible neighbors above or to the left of it, it is marked as inadmissible (if it hasn't been already marked as such) and no value is computed for it. If a cell has just one admissible neighbor above or to the left of it, only this value is used in previous formula. Otherwise, the algorithm proceeds filling the table exactly the same way as it's done in the section.



C



F



Possible paths

Tutorial 7

Dynamic Programing-2

CSC 311 (Fall 2018)

1- *Shortest-path counting* A chess rook can move horizontally or vertically to any square in the same row or in the same column of a chessboard. Find the number of shortest paths by which a rook can move from one corner of a chessboard to the diagonally opposite corner.The length of a path is measured by the number of squares it passes through, including the first and the last squares. Solve the problem by a dynamic programming algorithm.

With no loss of generality, we can assume that the rook is initially located in the lower left corner of a chessboard, whose rows and columns are numbered from 1 to 8 bottom up and left to right, respectively. Let $P(i, j)$ be the number of the rook's shortest paths from square (1,1) to square (i, j) in the ith row and the jth column, where $1 \leq i, j \leq 8$. Any such path will be composed of vertical and horizontal moves directed toward the goal. Obviously, $P(i, 1) = P(1, j) = 1$ for any $1 \leq i, j \leq 8$. In general, any shortest path to square (i, j) is reached either from its left neighbor, i.e., square $(i, j-1)$, or from its neighbors below, i.e., square $(i-1, j)$. Hence we have the following recurrence

$$\begin{aligned} P(i, j) &= P(i, j-1) + P(i-1, j) \text{ for } 1 < i, j \leq 8, \\ P(i, 1) &= P(1, j) = 1 \text{ for } 1 \leq i, j \leq 8. \end{aligned}$$

Using this recurrence, we can compute the values of $P(i, j)$ for each square $(i.j)$ of the board. This can be done either row by row, or column by column, or diagonal by diagonal. (One can also take advantage of the board's symmetry to make the computations only for the squares either on and above or on and below the board's main diagonal.) The results are given in the diagram below:

1	8	36	120	330	792	1716	3432
1	7	28	84	210	462	924	1716
1	6	21	56	126	252	462	792
1	5	15	35	70	126	210	330
1	4	10	20	35	56	84	120
1	3	6	10	15	21	28	36
1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1

2- *Rod-cutting problem* Design a dynamic programming algorithm for the following problem. Find the maximum total sale price that can be obtained by cutting a rod of n units long into integer-length pieces if the sale price of a piece i units long is p_i for i = 1, 2, . . . , n. What are the time and space efficiencies of your algorithm?

Let $F(n)$ be the maximum price for a given rod of length n. We have the following recurrence for its values:

$$F(n) = \max_{1 \le j \le n} \{p_j + F(n-j)\} \quad \text{for } n > 0,$$
$$F(0) = 0.$$

Using this recurrence, we can fill a one-dimensional array with $n+1$ consecutive values of F. The last value, $F(n)$, will be the maximum possible price in question. Since computing each value of $F(i)$ requires i additions (and i integer subtractions), the total number of additions is equal to $1 + 2 + \cdots + n = n(n+1)/2$, making the algorithm's time efficiency quadratic. The space efficiency of the algorithm is obviously linear since it uses an additional array of $n+1$ elements.

Actual cuts yielding the maximum price can be obtained by backtracing (see the examples in Section 8.1 and the solution to Problem 5 in these exercises).

(1)

ch 8 &U

Rod-cut

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

i	0	1	2	3	4	5	6	7	8	9	10
r[i]	0	1	5	8	10	13	17	18	22	25	30
s[i]	0	1	2	3	2	2	6	1	2	3	10

$i = 5$

$5 \longrightarrow 10$

$4+1 \longrightarrow 11$

$3+2 \longrightarrow 13$

$2+3 \longrightarrow (13)$

$i = 6$

$6 \longrightarrow 17$

$1+5 \longrightarrow 14$

$2+4 \longrightarrow 15$

$3+3 \longrightarrow 16$

7 ⟶ 17

1+6 ⟶ 18

2+5 ⟶ 18

3+4 ⟶ 18

---

8 ⟶ 20

1+7 ⟹ 19

2+6 ⟶ 22 ←

3+5 ⟶ 21

4+4 ⟶ 10

---

9 ⟶ 24

1+8 ⟶ 23

2+7 ⟶ 23

3+6 ⟶ 25

4+5 ⟶ 23

---

(9) إيجاد الحل الأمثل لتقطيع الـ

$S[9] = (3)$ → الجزء الأول (3) مقطع في الحل

$9 - 3 = 6$

$S[6] = (6)$ → الجزء الثاني مقطع في الحل

$9 = 3 + 6$

(6)

Rod cutting problem

مشكلة تقطيع لوح خشب

Length i	1	2	3	4	5	6	7	8	9	10
Price p_i	1	5	8	9	10	17	17	20	24	30

هذا المشكلة مقصود بها عندنا لوح خشب
مثلاً طوله (n) واريد ايجاد افضل
تقطيع يعطي اعلى سعر

مثلاً

length i = 5

price (p_i) = 10

r_i = 5+8 = 13

سعر قطعة طول 2 — سعر قطعة طول 3

سوف نستخدم الرموز التالية:

i ← طول القطعة

p_i ← سعر القطعة التالية

r_i ← افضل ربح من تقطيع القطعة التالية طولها i

(1)

طول القطعة	افضل تقطيع	الربح الناتج من افضل تقطيع
i = 1	1 = 1 no cut	$r_1 = 1$
i = 2	2 = 2 no cut	$r_2 = 5$
i = 3	3 = 3 no cut	$r_3 = 8$
i = 4	4 = 2+2	$r_4 = 5+5 = 10$
i = 5	5 = 3+2	$r_5 = 8+5 = 13$
i = 6	6 = 6 no cut	$r_6 = 17$
i = 7	7 = 2+2+3	$r_7 = r_4 + r_3 = 10+8 = 18$
i = 8	8 = 6+2	$r_8 = r_6 + r_2 = 17+5 = 22$
i = 9	9 = 6+3	$r_9 = r_6 + r_3 = 17+8 = 25$
i = 10	10 = 10 no cut	$r_{10} = p_{10} = 30$

القاعدة التي استخدمناها سابقاً.

$$r_n = \max\left(p_n, \underline{r_1 + r_{n-1}}, r_2 + r_{n-2}, \dots, \underline{r_{n-1} + r_1}\right)$$

مثلاً لو p_5 هطلب حساب r_5

نوجد أكبر max في كل ما يلي

$$r_0 + p_5 = 0 + 10 = 10$$

$$\left.\begin{aligned} r_1 + r_4 &= 1 + 10 = 11 \\ r_2 + r_3 &= 5 + 8 = 13 \\ r_3 + r_2 &= 8 + 5 = 13 \\ r_4 + r_1 &= 10 + 1 = 11 \end{aligned}\right\} \max = 13$$

$$r_5 = r_2 + r_3 = \boxed{13}$$

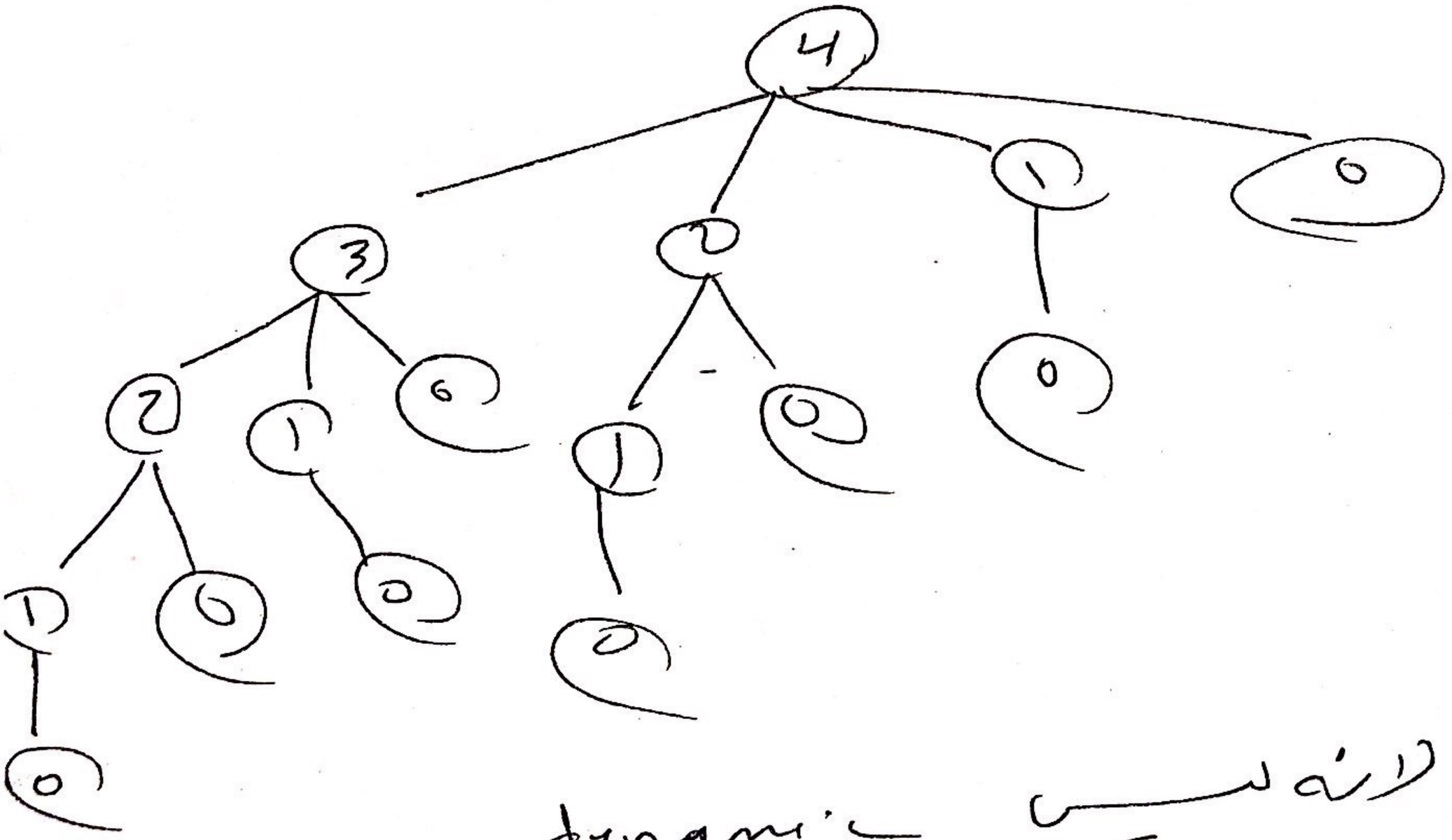
هذه القاعدة تسمى dynamic programming

القاعده [redacted] تشکل مختصر

$$r_n = \max_{1 \le i \le n} \left(p_i + r_{n-i} \right)$$

$$r_n = \max \left(p_1 + r_{n-1} \, , \, p_2 + r_{n-2} \, , \, p_3 + r_{n-3} \, , \, \ldots \, , \, p_n + r_0 \right)$$

شجره-ا ر مستدعا وا ی



هذا رابطه بین dynamic
میتوانید به تکرار کثیر